ORACLE
DEVELOP

S318534 - Building Maps with Oracle Spatial and Oracle Fusion Middleware MapViewer

Ji Yang
Principle Member of Technical Staff

Jayant Sharma
Technical Director

Joao Paiva
Principle Member of Technical Staff

Table of Contents

1. Launching MapViewer and running built-in demos

We will use the Oracle Fusion Middleware MapViewer Quickstart kit during this hands-on lab session. The Quickstart kit is available on MapViewer's OTN page. It has MapViewer pre-deployed into a standalone OC4J(Oracle Container for J2EE) instance, a lightweight J2EE application container.

The quick start kit is installed under C:\mvhol\mv1112_qs. All lab files are under C:\mvhol\mv1112_qs\oc4j\j2ee\home\default-web-app\hol, which the OC4J instance has direct access to. On the desktop of each student machine, we created shortcuts to script that launches OC4J and shortcuts to lab file folders for quick access.

During the lab session, we will use a free text editor, Notepad++, to edit our application code.

1.1 Launching OC4J

Let's first launch the standalone OC4J that comes with the Quickstart kit. This will also start MapViewer, which has already been deployed into the standalone OC4J.

1. Double click the shortcut "start_oc4j" on the desktop to open the home directory of the Quickstart kit. This shortcut points to start.bat under C:\mvhol\mv1112_qs, which has the following lines.

```
cd oc4j\j2ee\home
"C:\Program Files\Java\jdk1.6.0_21\bin\java" -server -Xmx384M -jar oc4j.jar
```

2. Wait until you see the following line at the bottom of the command window, which indicates that MapViewer is ready.

```
INFO: *** Oracle MapTileServer started. ***
```

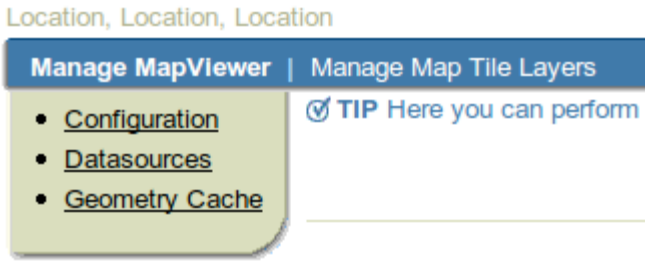
1.2 Verifying MapViewer is running properly

Now let's verify the MapViewer is running properly.

1. Launch the web browser by double clicking the shortcut to Firefox on the desktop.
2. Go to the MapViewer home page at the following URL.

   http://localhost:8888/mapviewer
3. Once the MapViewer home page is loaded, click the "Admin" button on the upper right side of the page.

4. On the Login page, type in “oc4jadmin” as the user name and “welcome” as the password. Click “Log In”.
5. Once logged into the MapViewer admin page, click the link “Datasources” on the left side of the page.



6. On the “Manage data sources” page, if you see two data sources, “mvdemo” and “hol2010”, then MapViewer is running properly and you're ready for the rest of the lab session.

   Note: A data source defines a database connection where MapViewer can get mapping data from. “mvdemo” and “hol2010” are two data sources that we have pre-configured for the lab. “mvdemo” connects to a local database user where the MapViewer standard demo data set is loaded, where is required to run the built-in demos and tutorials shipped with MapViewer. ”hol2010” connects to a database user where the mapping data for our hands-on lab application is loaded.
7. On the same page, click the link “Configuration” on the left side of the page.
8. The text box on this page displays the content of MapViewer XML configuration file, where all MapViewer configuration settings are defined. Scroll the text box all the way down to the bottom. You should see the two <map_data_source> tags that define the two data sources, “mvdemo” and “hol2010”. This is the place where you can add your own data source definition.

```
<map_data_source name="mvdemo"
                 jdbc_host="localhost"
                 jdbc_sid="orcl"
                 jdbc_port="1521"
                 jdbc_user="mvdemo"
                 jdbc_password="zA5R6pI4LeTmaIbgy4SbpczE3A6BBuRs"
                 jdbc_mode="thin"
                 number_of_mappers="3"
                 allow_jdbc_theme_based_foi="true"
 />

<map_data_source name="hol2010"
                 jdbc_host="localhost"
                 jdbc_sid="orcl"
                 jdbc_port="1521"
                 jdbc_user="hol2010"
                 jdbc_password="fJF6UbwGl3UOk4WRbp665eGxtoVTR8r7"
                 jdbc_mode="thin"
                 number_of_mappers="3"
                 allow_jdbc_theme_based_foi="false"
 />
```

1.3 Running the built-in demos

Now let's first take a look at a MapViewer demo called JView, which lets you write a SQL query and see its result on the map.

1. Click the "Demos" tab.

2. Click "JView" to launch the demo.
3. In the first text box, type in the following SQL query. This query selects all geometries from the states table.

```
Select geom from states
```

4. Click the "Submit" button. You should see a map displaying all states of US.

   Note: On the map, the visual result of a SQL query is called a theme. A map can have many themes.

5. In the second text box, type in the following SQL query. This query selects all geometries from the counties table whose state_abrv is 'CA'.

```
Select geom from counties where state_abrv='CA'
```

6. Click the box next to "Fill:" on the right hand side of the second text box. From the color selection widget, select green or some other color.
7. Click the "Submit" button. You should see a map displaying all counties in California in green on top of the states.

Next, let's take a look at some demos written with MapViewer's Javascript API, also known as Oracle Maps API.

1. Go back to the MapViewer home page, http://localhost:8888/mapviewer.
2. Click link "Oracle Maps Tutorial".
3. On the next page, click "Running the demos".
4. On the next page, you'll see 60+ Oracle Maps demos.
5. Launch demo No. 1 by clicking the link "Display Map".

   This demo shows a simple draggable map on the web page.

6. Drag the map around using the mouse. Zoom in or out the map by scrolling the mouse wheel.
7. Launch demo No. 5 by clicking the link "Theme Based FOI layer visibility".

   This demo displays two interactive feature layers/themes on the map. The theme with red dots shows the customer locations. The theme with colored polygons shows counties with different

population. Different colors indicate different population.

8. Click the red dots or the colored county polygons to view additional feature attributes.
9. Turn on/off a feature layer(theme) by chcking/unchecking the check boxes.

2. Running the lab application and adding a theme.

From this lab, we will start building our own Oracle Maps application. In the lab, we will start with a very basic Oracle Maps application.

2.1 The starter application

Let's take a look at the starter application, which can be accessed at http://localhost:8888/hol/lab2 in the web browser. The application displays a map, which has a Navigation panel and a tool bar. The tool bar has control buttons for two built-in tools, distance measurement tool and marquee zoom tool. You can activate the tools by clicking the corresponding control buttons.

If you look carefully, you'll see the Google logo at the lower-left corner of the map. That tells us that the map displayed by this application is Google Maps. In addition to Google Maps, Oracle Maps Javascript API also lets you display Microsoft Bing Maps and Yahoo Maps as the background map. Of course, you can always use your own background maps served by MapViewer, as we've just seen with the Oracle Maps demos in Lab 1.

Now double click the shortcut to "lab2" on the desktop to open the folder that contains the application files. This folder has the following files/folder.

- index.html

  This is the HTML page of the application. It displays the application header and a map area beneath it.

- page.css

  The CSS file that defines various styles used in the application web page.

- images

  This folder contains the logo image displayed as part of the header on the application web page.

- create_view.sql

  Strictly speaking, this file is not really part of the application, because it is not executed when the application is running. It has already been executed to create a database view, on which a theme query is based on. We'll take a closer look at this file next.

- application.js

  The Javascript file that implements the application logic. It has two functions.

  - init

    This function is called when the application web page is loaded into the browser. It performs the following tasks.

    - Create a Oracle Maps client instance,
    - Add the Google Maps layer,
    - Add the navigation panel.
    - Call addTooBar.

- Set the initial map zoom level.
- Display the map.

- addToolBar

  This function adds the tool bar to the map.

2.2 Changing background map

The background map of an Oracle Maps application is displayed as a map tile layer. Every map must have at least one map tile layer. The map tile layer can be generated by MapViewer using your own map data. It can also come from popular map services such as Google Maps, Microsoft Bing Maps and Yahoo Maps. In this part of the lab, we will change our application to use different map tile layer.

1. Open the file application.js under the folder lab2 using Notepad++.
2. Comment line #10.

```
//var baseMap = new MVGoogleTileLayer() ;
```

3. Uncomment line #12 and 13.

```
var baseMap = new MVMapTileLayer("elocation_mercator.world_map",
"http://elocation.oracle.com/mapviewer/mcserver") ;
```

4. Save the file and refresh the application page in web browser. The background map should be changed to Oracle eLocation map.
5. Comment line #12 and 13.
6. Uncomment line #16.

```
var baseMap = new MVBingTileLayer() ;
```

7. Because Bing map does not support wraparound map mode, you also need to comment line #20.

```
//mapview.enableMapWrapAround(true) ;
```

8. Save the file and refresh the application page in web browser. The background map should be changed to Microsoft Bing map.

2.3 Defining a theme

In this part of the lab, we'll define a theme that displays customers as markers on the map.

A theme on the map is the visual representation of a database SQL query that returns spatial/geographical attributes. Defining theme is basically the process of defining the query and the rules to style the geographical features.

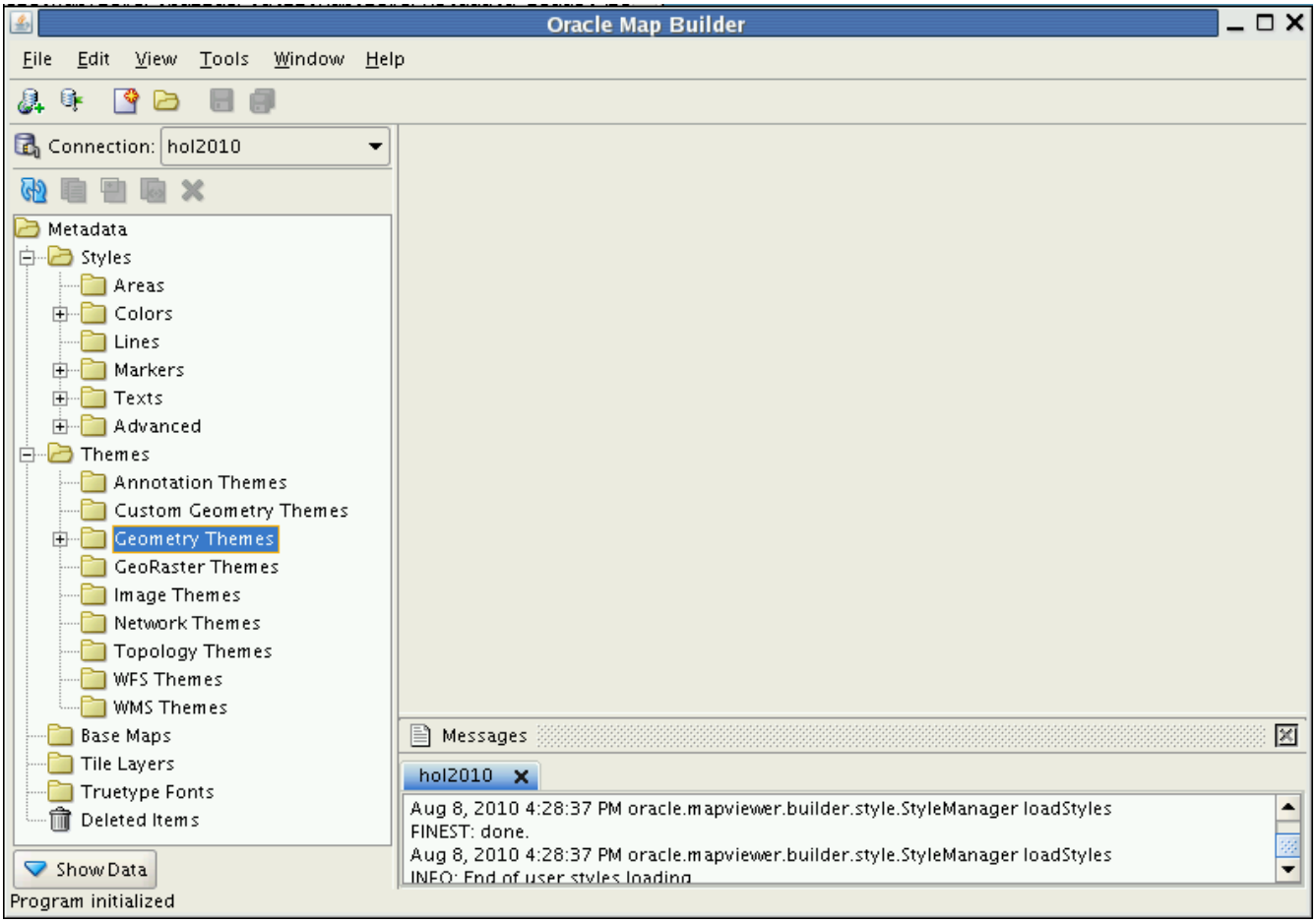
In the simplest cases, the query is based on a single table, which stores all geographical and non-geographical attribute data that the theme needs. In more complicated cases, the theme data needs to come from multiple tables and thus the query has to be based on multiple tables or a view joining multiple tables.

The data of our customer theme comes from two tables, SH_CUSTOMERS and SH_SALES. SH_CUSTOMERS has basic customer information such as name, id, address, location, income level and other demographic attributes. SH_SALES has records of the purchases that the customers have made in the past. We created a view named CUSTOMERS joining these two tables so that our theme has basic customer information plus the total amount of money that the customers have spent. You can open the SQL script create_view.sql under the lab2 folder and view what it does. Later on, you'll see how we're going to use the money amount in our application.

Theme definition is stored in the database. MapViewer can read the theme definition through the database connections defined by its data sources. You can use Map Builder, a desktop tool shipped with MapViewer, to define themes and maps and write the definition to the database.

Now let's define the theme using Map Builder.

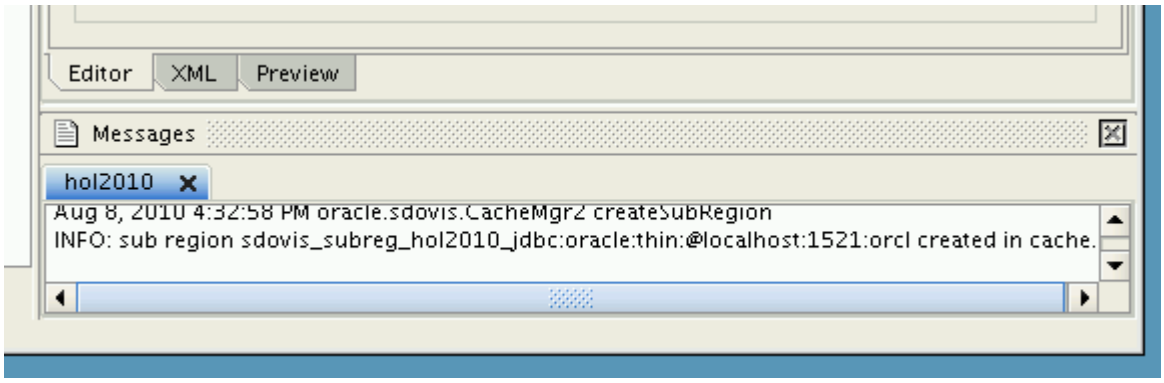
1. Double click the shortcut “MapBuilder” on the desktop to launch Map Builder.
2. Select “Load/Add/Remove” from the dropdown list next to “Connection:”.
3. In the dialog window, select “hol2010” and click button “Load”. This connects Map Builder to the database user where our lab data is loaded.
4. In the Metadata tree list inside the left panel, expand the “Themes” node by clicking on the “+” sign next to the “Themes” node or double clicking the “Themes” node.



5. Expand the "Geometry Themes" node.
6. Right click on the "Geometry Themes" node and click "Create Geometry Theme" on the popup menu.
7. Click next on the "Define a Geometry Theme" wizard window.
8. On the next page, change Name to "customers". Select "HOL2010" from the Table Owner drop-down list. Select "CUSTOMERS" from the Base Table drop-down list. Select "LOC3785" from the Spatial Column drop-down list. Click Next.
9. On the next page, select "Marker" from the Style Type drop-down list.
10. In the Style Picker Dialog, select "M.STOPLIGHT_RED" from the list. Click Ok.
11. Click Next.
12. On the "Step 3 of 4 – Style" page, check "Label Style". Click the field below "Attribute", which should be "AMOUNT_SOLD", and select "CUST_ID" from the drop-down list. Click "Next".
13. On the "Step 4 of 4 - Query Condition" page, click "Next".
14. On the "Completed" page, click "Finish".
15. Click "Advanced" inside the "Theme Options" panel. Change Key Column from "ROWID" to

"CUST_ID".

16. Save the changes by clicking the disk icon on the tool bar.
17. Now let's take a quick look at the theme. Click the "Preview" tab above the "Messages" panel at the bottom of the Map Builder window.



18. Click the play button . You should see the customers being rendered as small red circle markers.

2.4 Adding the customers theme to the application

In this part of the lab, we will display the customers theme on our application's map.

1. Open application.js in folder lab2 with Notepad++.
2. Open lab2.txt on the desktop.
3. Copy the following line from lab2.txt and paste it to application.js at line #24.

```
addCustomerTheme();
```

4. Copy the following function from lab2.txt and paste it at the end of application.js.

```
/** This function adds a map layer that displays the customers as markers.
 */
function addCustomerTheme()
{
  theme = new MVThemeBasedFOI("customers", "hol2010.CUSTOMERS") ;
  theme.setQueryWindowMultiplier(1) ;
  mapview.addThemeBasedFOI(theme) ;
}
```

5. Save application.js.
6. Refresh the application page http://localhost:8888/hol/lab2 in the browser. The application page should show the customers theme as red markers on the map.

3. Improving the theme with information window and advanced style

Our application can now display the customers on the map. But it is still not very informative. Other than the location, there is nothing else we can learn about the customers from the map. In this lab, we will improve our map and make it more informative by doing the following two things.

1. We will use markers of different colors to represent customers with different purchase amount. When the user looks at the map, he or she will be able to tell roughly how much the customers have spent in addition to their geographic location.
2. We will add information window to the customers theme. When the user clicks on a customer on the map, the information window will popup and it will shows some basic information about the customer, such as name, id, country, income level and etc.

Before we start adding things, let's first take a look at the starter application for this lab. Open the application by going to http://localhost:8888/hol/lab3 in the browser.

As we can see, this application looks almost the same as the one we built in lab 2. The only difference is that the new start application shows a map legend at the lower-right corner of the map. It shows the three different colors that we're going to use to represent three different ranges of amount of money that the customers have spent.

Now let's improve the customers theme step by step.

3.1 Replacing the simple marker style with an advanced style

In this part of the lab, we will replace the current simple marker style of the customers theme with an advanced style. This advanced style lets MapViewer choose one from three different markers for each customer based on the total amount of money the customer has spent.

Let's first take a look at the advanced style that we will use for the customers theme.

1. Open Map Builder. If Map Builder has been shutdown, start it and connect it to the datasource "hol2010".
2. In the "Metadata" tree list, expand the "Styles" node by clicking the "+" next to "Style".
3. Expand the "Advanced" node.
4. Double click "V.MARKER_SALES".
5. The main panel now shows the definition of the advanced style named "V.MARKER_SALES".

   The "Bucket Information" panel shows three entries. Each entry defines a numerical range with Low bound and High bound, and the marker style used for the range. It also specifies the display name of the range when it is displayed in a map legend.

Now let's change the customers theme.

1. In the “Metadata” tree list, expand “Themes”->”Geometry Themes”.
2. Double click “CUSTOMERS” to open the theme definition.
3. Switch back to the “Editor” view if Map Builder shows the “Preview” mode. To switch back to the “Editor” view, click the “Editor” tab above the “Messages” panel at the bottom of the Map Builder window.
4. Click “Styling Rules” inside the “Theme Options” panel.
5. Inside the “Styling Rules” panel, select the only rule entry and click  .
6. In the “Edit Styling Rule” dialog window, click  next to the “Attribute Columns” text field.
7. Select “AMOUNT_SOLD” from the list on the left and click  to add it to the list on the right side.
8. Click Ok.
9. Click  next to the “Render style” text field.
10. In the “Style Picker Dialog” window, select “ADVANCED” from the “Style Type” drop-down list.
11. Select “V.MARKER_SALES”.
12. Click Ok.
13. Click Ok.
14. Save the changes by clicking the disk icon on the tool bar.
15. Now let's take a quick look at the theme. Click the “Preview” tab above the “Messages” panel at the bottom of the Map Builder window.
16. Click the play button(green triangle). You should see the customers being rendered as markers of different colors.

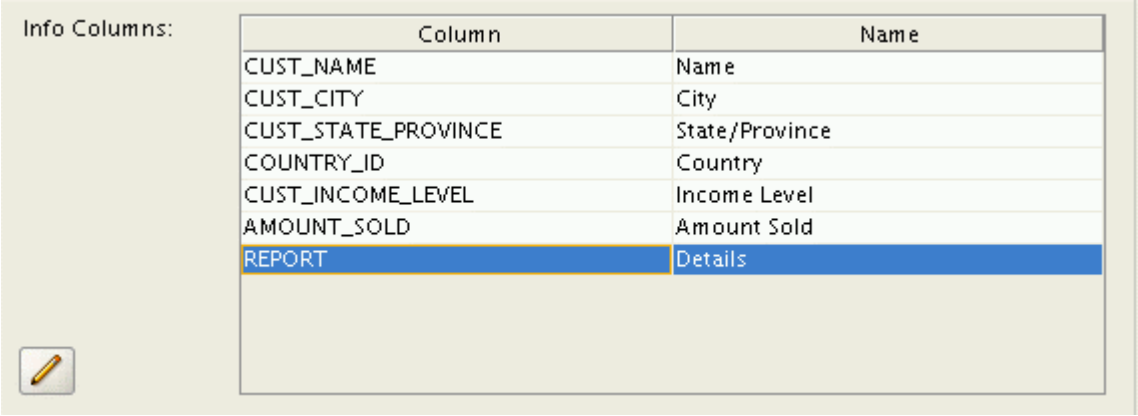
3.2 Specifying information window attributes

In this part of the lab, we will specify several attributes that will be displayed inside the popup info window displayed when the user clicks the customers on the map. The order in which the information attributes are defined affects how the application functions. So please strictly follow the order in which the attributes are added as described next.

1. In Map Builder, switch back to the “Editor” mode from the “Preview” mode by clicking the “Editor” tab above the “Messages” panel at the bottom of the Map Builder window.
2. Click “Advanced” inside the “Theme Options” panel.
3. Inside the “Advanced Parameters” panel, click  at the lower-left corner.
4. In the “Edit Info Columns” dialog window, click  to add a new information attribute.
5. The default attribute added is “AMOUNT_SOLD”. Click on it and select “CUST_NAME”

from the drop down list. Double click the emtpy field next to “CUST_NAME” and type in “Name”, which will used as the display name of the attribute in the information window.

6. Repeat 4 and 5. Add “CUST_CITY” as an information attribute and set its display name to “City”.
7. Repeat 4 and 5. Add “CUST_STATE_PROVINCE” as an information attribute and set its display name to “State/Province”.
8. Repeat 4 and 5. Add “COUNTRY_ID” as an information attribute and set its display name to “Country”.
9. Repeat 4 and 5. Add “CUST_INCOME_LEVEL” as an information attribute and set its display name to “Income Level”.
10. Repeat 4 and 5. Add “AMOUNT_SOLD” as an information attribute and set its display name to “Amount Sold”.
11. Repeat 4 and 5. Add “REPORT” as an information attribute and set its display name to “Details”.



12. After verifying the “Edit Info Columns” dialog has the above content. Click “Ok”.
13. Save the changes by clicking the disk icon on the tool bar.

3.3 Clearing map meta-data cache

Before we can see the theme definition changes in our application, we must clear MapViewer's metadata cache, which stores the style, theme and map definition that it has loaded previously from the database. If we fail to do it, MapViewer will hold on to the old theme definition and will not pick up the changes that we've just made using Map Builder.

So let's clear MapViewer's metadata cache.

1. Go to MapViewer's home page http://localhost:8888/mapviewer. in the browser.
2. Click the “Admin” button on the upper right side of the MapViewer home page.
3. Since you've already logged in once, you shouldn't be asked for username/password this time. If you're asked to log in again, type in “oc4jadmin” as the user name and “welcome” as the password. Click “Log In”.

4. Once logged into the MapViewer admin page, click the link “Datasources” on the left side of the page.
5. On the “Manage data sources” page, select the radio button of data source “hol2010” and click button “Purge cached metadata”.
6. You will see information that indicates that the cached mapping meta-data has been cleared.
7. Test the application. Go to http://localhost:8888/hol/lab3 in the browser and refresh the page.
8. Once the page is refreshed, you should see the customers are displayed in different colors based on the AMOUNT_SOLD attribute. Clicking a customer on the map should bring up a popup window showing the values of the attributes that you've just defined using Map Builder. On the bottom of the information window, the value field of “Details” always shows a link “Click here”. But when you click the link, nothing happens. We'll fix this issue next.

3.4 Showing detailed customer report

The name of the last info column that you added using Map Builder is “REPORT” and we set its display name to “Details”. We can see how the column “REPORT” is defined for the view CUSTOMERS in file create_view.sql under the lab2 folder.

```
create or replace view customers as
select
    a.CUST_ID,
    ...
    amount_sold,
    '<a href="javascript:showReport(' || a.CUST_ID || ')">Click here</a>' REPORT
  from
    ...
```

The value of the column “REPORT” is a string that specifies the following HTML hyper link tag.

```
<a href="javascript:showReport(<cust_id>)">Click here</a>
```

When being displayed in the information window, it shows a link named “Click here”. When being clicked, the link invokes a Javascript function named “showReport”, with the customer ID being passed as the parameter.

Because we haven't defined the function “showReport” for our application, nothing happens when you click the “Click here” link. Next we will fix this problem by adding the function to our application. After that our application will be able to show a detailed customer report when the user clicks on the “Click here” link.

1. Open the file lab3.txt on the desktop.
2. Click the shortcut to folder lab3 on desktop to open the application files folder.
3. Open application.js with Notepad++.
4. Copy the definition of function showReport from lab3.txt and paste it at the end of

application.js.

```
function showReport(customerId)
{
  window.open('report.jsp?cust_id='+customerId,
              'Customer Report',
              'width=600,height=500,resizable=yes,scrollbars=yes');
}
```

The showReport function opens report.jsp with the customer ID as a parameter inside a new browser window. This JSP page queries the database and outputs detailed customer information and the full history of all purchases that the customer has made.

5. Save application.js.
6. Go back to the application, http://localhost:8888/hol/lab3, in the browser and refresh the page.
7. Click on a customer and then click "Click here" in the information window. You should see a new browser window showing the detailed customer report.

4. Adding selective filters

In this lab, we will further enhance our application by adding selective filters to the customers theme. The filters are basically query conditions added to the customers theme query. With the filters, the user will be able to change the result of customers theme query based on query criteria chosen on the fly.

Let's first take a look at the starter application of this lab, http://localhost:8888/hol/lab4.

Compared to the finished product of lab 3, the only thing that we've added to the starter application is a panel on the left that lists three filters, income level, country and state/province. At this point, the three filters are nothing more than three empty select lists, which are added with the following HTML code in index.html.

```
<tr>
   <td>Income Level</td>
   <td><select id="income_level_list" onchange="filterChanged(this)"/></td>
</tr>
<tr>
   <td>Country</td>
   <td><select id="country_list" onchange="filterChanged(this)"/></td>
</tr>
<tr>
   <td>State/Province</td>
   <td><select id="province_list" onchange="filterChanged(this)"/></td>
</tr>
```

In the rest of this lab, we will add code to populate these select lists with options dynamically gathered from the theme data. We will also add code to issue new theme queries with modified query conditions when the user changes the filter selections.

4.1 Adding dynamic query conditions to the theme

In this part of the lab, we will add three dynamic query conditions to the customers theme. With these dynamic query conditions, we will be able to change the theme query result on the fly based on the user selections.

1. Open lab4.txt on the desktop.
2. Open Map Builder. If Map Builder has been shutdown, start it and connect it to the datasource "hol2010".
3. In the "Metadata" tree list, expand "Themes"->"Geometry Themes".
4. Double click "CUSTOMERS" to open the theme definition.
5. Switch back to the "Editor" mode if Map Builder shows the "Preview" mode. To switch back to the "Editor" view, click the "Editor" tab above the "Messages" panel at the bottom of the Map Builder window.
6. Click "Styling Rules" inside the "Theme Options" panel.
7. Inside the "Styling Rules" panel, select the only rule entry and click  .

8. In the “Edit Styling Rule” dialog window, copy the following query conditions from lab4.txt and paste them into the “Query Condition” text box. These query conditions are added to the SQL query WHERE clause. The values of the three binding variables supplied during runtime will determine the result of the query.

```
cust_income_level like :1 and
country_id like :2 and
cust_state_province like :3
```

9. Click Ok.
10. Save the change by clicking the disk icon on the tool bar.
11. Click the “Preview” tab above the “Messages” panel at the bottom of the Map Builder window.
12. Click the play button(green triangle).
13. You should see a window pops up and asks you to provide the values of the three binding variables defined in the query conditions. Type in the wild card symbol, “%”, for all three binding variables.
14. Click Ok.
15. The map should display all customers.

4.2 Clearing map meta-data cache and setting query binding variables

In order to make the changes effective, we must clear MapViewer's metadata cache.

1. Go to MapViewer's home page http://localhost:8888/mapviewer. in the browser.
2. Click the “Admin” button on the upper right side of the MapViewer home page.
3. Since you've already logged in once, you shouldn't be asked for username/password this time. If you're asked to log in again, type in “oc4jadmin” as the user name and “welcome” as the password. Click “Log In”.
4. Once logged into the MapViewer admin page, click the link “Datasources” on the left side of the page.
5. On the “Manage data sources” page, select the radio button of data source “hol2010” and click button “Purge cached metadata”.
6. You will see information that indicates that the cached mapping meta-data has been cleared.
7. Test the application. Go to http://localhost:8888/hol/lab4 in the browser and refresh the page.
8. Once the page is refreshed, you will see that the customers are all gone and the clock in the middle of the map ticks forever! That's because our application is not supplying the query binding variables required by the customers theme. Let's fix it next.
9. Open the lab4 file folder by double clicking the shortcut to lab4 on the desktop.
10. Open application.js in the lab4 folder with a Notepad++.

11. Copy the following line from lab4.txt and paste it to application.js at line #38. Just like what we did when previewing the theme in Map Builder, this line supplies the wild card symbol as the binding variables, which in effect will return all customers.

```
theme.setQueryParameters("%", "%", "%") ;
```

12. Save application.js.
13. Refresh the application page in the web browser. You should see that the customers are back on the map.

4.3 Populating filter select lists

In this part of the lab, we will add code to populate the select lists with data returned by the theme query.

1. Open application.js in the lab4 folder with Notepad++.
2. Copy the following line from lab4.txt and paste it into application.js right below the line that we have just added.

```
theme.attachEventListener(MVEvent.AFTER_REFRESH, populateFilterLists) ;
```

   The above line adds a AFTER_REFRESH listener to the theme. The AFTER_REFRESH listener is a Javascript function that is invoked after the theme query result is returned from the server and the theme has been displayed on the map.

   The function "populateFilterLists" has already been defined in the file categories.js in the lab4 folder. It goes through all customers data records returned from the server and collects information such as the unique income levels, countries and states/provinces, which the customers belong to. It then populates the three select lists with the these values.

3. Save application.js.
4. Refresh the application page in the web browser. Now the income level and country select lists should be populated with values returned from the theme query.

You can change the filter selections, but it won't change the theme on the map. That's because we haven't implemented the code that issues a new theme query when the filter selections are changed.

4.4 Redrawing the theme with changed filter values

Remember the HTML code that adds the select lists to the application page?

```
<select id="country_list" onchange="filterChanged(this)"/>
```

For each select list, we specified an onchange listener function, "filterChanged". This function is invoked whenever the user chooses a different value from the dropdown select list. It sets the theme query binding variables based on the user's selections and refreshes the theme with a new query. For the country select list, it will also update the State/Province list with the list of states/provinces that the

selected country has.

Now let's add the function "filterChanged" to our application to complete our application.

1. Open application.js in the lab4 folder with Notepad++.
2. Copy the following lines from lab4.txt and paste it at the end of application.js.

```
function filterChanged(target)
{
  /** Get the selected option values from the select lists */
  var country = getSelectedValue("country_list") ;
  var incomeLevel = getSelectedValue("income_level_list") ;
  var province = getSelectedValue("province_list") ;
  if(country=="%")
    province = "%" ;

  /** If the country selection is changed, we should also
   *  popluate the state/province select list with the list
   *  of the states/provinces of the selected country.
   **/
  if(target.id == "country_list")
  {
    var provinces = countries[country] ;
    populateSelectList("province_list", provinces) ;
  }

  /** Set the theme binding variables with selected values and
   *  refresh the theme.
   **/
  theme.setQueryParameters(incomeLevel, country, province) ;
  theme.refresh() ;
}
```

   The above code defines the function "filterChanged", which is invoked whenever the user changes the selected filter values. It does the following things.

   - Get the selected filter values from the select lists.
   - If the changed select list is the country list, populate the state/province select list with the selected country's states/provinces.
   - Sets the theme binding variables with selected filter values and refreshes the theme.

     ```
     theme.setQueryParameters(incomeLevel, country, province) ;
     theme.refresh() ;
     ```

3. Save application.js.
4. Refresh the application page in the web browser. The application should be fully functional.